Studying the Impact of Developer Communication on the Quality and Evolution of a Software System
A Doctoral Dissertation Retrospective

Nicolas Bettenburg
Software Analysis and Intelligence Lab (SAIL)
School of Computing
Queen’s University
Kingston, ON, CANADA
Email: nicbet@cs.queensu.ca

Abstract—Software development is a largely collaborative effort, of which the actual encoding of program logic in source code is a relatively small part. Software developers have to collaborate effectively and communicate with their peers in order to avoid coordination problems. To date, little is known how developer communication during software development activities impacts the quality and evolution of a software. In the doctoral thesis presented to this symposium, we introduce and evaluate tools and techniques to recover communication data from traces of the software development activities that are recorded in software repositories. Using this data, we present a study on the impact of developer communication on the quality and evolution of the software through a tiered study.

Among the highlights of the presented dissertation, we find that communication between developers stands in a direct relationship to the quality of the software. In addition, we find that communication between developers plays a important role in maintaining a healthy contribution management process, which is one of the key factors to the successful evolution of the software. Our dissertation illustrates that software development is an intricate and complex process, which is strongly influenced by the social interactions between the stakeholders involved in the development activities. A traditional view based solely on technical aspects of software development such as source code size and complexity, while valuable, limits our understanding of software development activities. The presented research consists of a first step towards gaining a more holistic view on software development activities.

I. INTRODUCTION

Software development is a highly complex task. In order to cope with this complex task, development effort is often split across individuals or teams, who are responsible for one or more (less complex) concerns of the development effort [2], [15]. Managers often use high-level information about the software system to divide the development effort into work teams [6]. However, such a division of the development effort increases the need for coordination and communication among developers [14].

Melvin Conway in 1968 informally described this duality between modularization of the source code and the organization of developers responsible for creating that source code as Conway’s Law. The source code of a software system stands in direct relation with the organizational structure of the development team, as design decisions require communication among the stakeholders making those decisions [8].

Based on that need for communication and coordination, it comes to no surprise that software developers typically spend a large fraction of their work day communicating with their co-workers [3]. For instance, Wu et al. report in an observational study on a large industrial software development company that engineers spent up to 2 hours each day communicating with their co-workers [20].

Recent research suggests that communication has a direct impact on the software development efforts. For instance, de Souza et al. provided evidence that communication among software developers is a critical factor for the success of software development efforts: information hiding led development teams to be unaware of other teams’ work, resulting in coordination problems [9]. Similar findings for distributed software development have been reported by Bird et al. and Grinter et al. [4], [12], who highlight communication as the most referenced problem in distributed software development [4].

Lately, research in Empirical Software Engineering has begun to understand that the multitude of social and interpersonal interactions that go hand-in-hand with communication may also impact the software development process, as well as the software product itself. For instance, research has demonstrated that the forming of social networks and sub-communities [5], [19], work dependencies [7], or daily routines [17] stand in a direct relation to the quality of the final software product [6].

II. DISSERTATION HYPOTHESIS

Motivated by the empirical evidence provided in past research, this dissertation proposes to study the impact of communication between stakeholders in the software development process, such as developers and users, on the quality of the software. We believe that the quality of a software system, is impacted not only by the condition of the source code, but also by the collaborative activities of the stakeholders responsible for the development of the software system.

These beliefs led us to the formulation of our research hypothesis, which we state as follows:

The communication between software developers plays a key role in the quality and the evolution of the software.
Communication between stakeholders can materialize in a broad spectrum of activities. As our research hypothesis is formulated within a broad context, we focus on two particular manifestations of developer communication, for which we are able to extract and synthesize data from the development repositories that report the day-to-day software development activities. For this purpose, we formulate two sub-hypotheses:

1) The communication between software developers stands in a direct relationship to software quality.
2) The communication between software developers stands in a direct relationship to software evolution.

First, we develop tools and techniques to extract communication data from development repositories, and empirically validate these tools and techniques through case studies on development repositories of several open-source software projects. Next, we perform a linking of the extracted communication data back to the parts of the software system referenced within the data. These links enable us to investigate the impact of communication between stakeholders on software quality.

Second, we conjecture that a software system evolves through daily development activities that are captured on a technical level as changes (called patches) to the existing source code. Communication between developers materializes in the way through which this daily development effort is managed. To validate our research hypothesis, we propose to extract information about the management of code contributions, such as bug fixes, patches, or new features. We then derive a formal model of contribution management from a broad spectrum of open-source software systems. Based on this model, we perform an in-depth case study of two successful, large-scale open-source software systems. By contrasting management practices and processes, we are able to demonstrate that the management of contributions has a direct impact on the future inflow and integration of development effort into the software, and thus directly impacts the evolution of a software system.

III. DISSERTATION OVERVIEW

In the following, we present an overview of the organization of our doctoral dissertation. In particular, our dissertation consists of five main chapters. Each chapter is dedicated towards a specific research problem. Our goal was to compose each chapter as a self-contained unit, such that readers can peruse each chapter independently. We have organized the chapters of this dissertation into four parts.

Part I of our dissertation introduces the research problem, as well as our research hypotheses (Chapter 1). In addition, we present a broad background on the research topic, as well as a literature review and in-depth discussion of work related to this dissertation in Chapter 2.

Part II of our dissertation presents tools and techniques for mining collaboration data from software development repositories. In particular, we present a detailed discussion of the perils and pitfalls of mining communication data from e-mail communication archives (Chapter 3), as well as tools and techniques for separating natural language text from technical information (e.g., class names) contained in communication data (Chapter 4). In Chapter 5, we demonstrate through an empirical study that we can use the proposed tools and techniques to link communication data surrounding the software development process to the software product itself (i.e., the source code).

Part III of our dissertation is dedicated to the empirical study of our hypothesis, using implementations of the tools and techniques form Part II. In Chapter 6, we use these links to explore the first part of our research hypothesis. In particular, we demonstrate that communication, among other social properties of development teams, has a strong relationship to software quality. In Chapter 7, we present an empirical study on how developer communication surrounding the management of code contributions impacts the evolution of a software system.

Part IV of our dissertation provides an outline of future research directions in Chapter 8, together with a summary of the main contributions of our dissertation, as well as our concluding remarks.

IV. DISSERTATION SUMMARY

Source code is the end-product of a variety of collaborative activities, carried out by the developers of a software. Lately, research has begun to understand that the intricacies of these activities, such as social networks, work dependencies, or daily routines, stand in a direct relation to the successful evolution of the software. Traces of these collaborative activities are captured throughout the development of the software and can be found in a large variety of repositories that developers use on a day-to-day basis, such as issue tracking systems, email communication archives, or version archives.

In this dissertation, we present tools and methods to mine communication data, as an artifact describing the collaborative efforts of software developers. We use the extracted information to empirically study the relationship between communication and the evolution of the software. We study two manifestations of developer communication. First we study developer communication that is focussed on issue reports such as bug reports. Second, we study developer communication that is focussed on source code contributed by peer developers. The aim of our work is to feedback the insights gained from our empirical studies, to assist stakeholders in making future decisions and ultimately increasing software quality, as well as development efficiency.

In the following we provide short summaries of each chapter of our doctoral dissertation, together with a summary of the highlights of each chapter.

Chapter 2 - Background and Literature Review

Past research has produced an extensive body of work describing relationships between socio-technical information about the software development process, the stakeholders surrounding software development activities, and the software product. This body of work consists of a multitude of unique data sources, data mining techniques, social and technical
metrics, as well as a diverse set of modeling strategies, evaluations and recommendations to practitioners. In this chapter, we present a systematic survey of existing literature on the use of socio-technical information in software quality assurance, to compare and contrast prior work in this emerging research field, and to identify open research opportunities.

Our systematic literature summarizes research that relates socio-technical information about software development efforts to software quality. We identify and critically disseminate a total of 37 primary studies from a broad range of software engineering venues. We uncover a total of nine distinct research topics that are covered in existing literature, studying a broad range of relationships between socio-technical concerns and software quality. In particular we find a total of 39 different technical, social, and socio-technical metrics described in literature, together with investigations on their relationships to a total of 13 different quality metrics.

Within each research topic, we present meta-analyses that summarize and outline the main findings within each topic, and we identify future research opportunities in the area.

Chapter 3 - Mining Communication Data

The Collins English Dictionary defines collaboration as "... working with each other to do a task and to achieve shared goals [16]." A central aspect of collaboration is the communication required to coordinate work. A recent study in the domain of software engineering shows that engineers spend up to 2 hours each day communicating across various channels (face-to-face, chat, email) to coordinate their software development efforts [20]. As a result, communication repositories, such as mailing lists, contain valuable information about the history of a software project. Research is starting to mine this information to support developers and maintainers of long-lived software projects. However, such information exists as unstructured data that needs special processing before it can be studied.

In this chapter, we identify several challenges that arise when using off-the-shelf techniques for mining mailing list data. In addition, we evaluate the negative impact of several of these mining challenges on the final data and research results, and we propose solutions to effectively tackle these mining challenges. We have implemented a publicly available tool for mining mailing list repositories, which has already found extensive use in the research domain [11], [13], [18].

Chapter 4 - Mining Technical Information from Communication Data

Communication data mined from email, chat, or issue report comments, frequently consists of unstructured data, i.e., natural language text, mixed with technical information such as project-specific jargon, abbreviations, source code patches, stack traces and identifiers. Technical artifacts represent a valuable source of knowledge about the software. The intertwining between natural language and technical content make the separation of these two types of text challenging.

In this chapter, we present a general-purpose, yet lightweight approach to extracting technical information from unstructured data. Our approach is based on existing spell checking tools, which are well-understood, fast, readily available across platforms and impartial to different kinds of technical artifacts. Through a handcrafted benchmark, we demonstrate that our approach is able to successfully uncover a wide range of technical information in unstructured data, and provides a statistically significant improvement over the state-of-the-art (+23% precision, +16% recall). In future work, we plan to study the use of additional heuristics to increase recall and carry out a more detailed evaluation on different kinds of technical information through an extended benchmark.

Chapter 5 - Linking Communication Data to Source Code

When discussing software, practitioners often reference parts of the project’s source code. In the previous chapter, we have presented a lightweight approach to separating technical information such as class names, function names, stack trace, or source code examples from natural language text. Such references have different motivations, such as mentoring and guiding less experienced developers, pointing out code that needs changes, or proposing possible strategies for the implementation of future changes. Knowing which code is being talked about the most can not only help practitioners to guide important software engineering and maintenance activities, but also act as a high-level documentation of development activities for managers.

In this chapter, we present an approach based on clone-detection as specific instance of a code search based approach for establishing links between code fragments that are discussed by developers and the actual source code of a project. Through a case study on the Eclipse project, we explore the traceability links established through this approach, both quantitatively and qualitatively, and compare our approach to classical linking approaches, in particular change log analysis and information retrieval. The results of our study show that the links established through fuzzy code search are conceptually different than traditional approaches based on change log analysis or information retrieval.

In the future, we envision traceability links established by our approach to be used to assist practitioners when browsing issue reports. A sample application would be an enhanced BugZilla system, which assists the bug fixing process by identifying code fragments contained in the corresponding issue report discussion and points developers to the locations in the project’s source code that contain similar or identical code.

Chapter 6 - The Relationships between Communication and Software Quality

Correcting software defects accounts for a significant amount of resources in a software project. To make best use of testing efforts, researchers have studied statistical models to understand in which parts of a software system future defects are likely to occur. By studying the mathematical relations between explanatory variables used in these models, researchers can form an increased understanding of the important connections between development activities and software quality. Explanatory variables used in past top-performing models are largely based on source code-oriented metrics, such as lines of code or number of changes. However, source code is the
end product of numerous interlaced and collaborative activities carried out by developers. Traces of such activities can be found in the various repositories used to manage development efforts.

In this chapter, we present statistical models to study the impact of social interactions in a software project on software quality. These models use explanatory variables based on social information mined from the issue tracking and version control repositories of two major open-source software projects. The results of our case studies demonstrate the impact of metrics from four different dimensions of social interaction on post-release defects. Our findings show that statistical models based on social information have a similar degree of explanatory power as traditional models. Furthermore, our results demonstrate that social information does not substitute, but rather augments traditional source code-based metrics used in explaining software defects.

Chapter 7 - The Impact of Communication on the Evolution of the Software

In recent years, many companies have realized that collaboration with a thriving user or developer community is a major factor in creating innovative technology driven by market demand. As a result, businesses have sought ways to stimulate source code contributions from developers outside their corporate walls, and integrate external developers into their development process. The communication between volunteer developers from the community and core developers of the software is a central part of how external source code contributions are managed and integrated into the software.

In this chapter, we investigate developer communication surrounding source code contributions. In particular, we investigate this relationship through an empirical study on the contribution management of two major, successful, open source software ecosystems (Android and the Linux kernel). We base our analysis on a conceptual model of contribution management that we derived from a total of seven major open-source software systems. This model defines five phases of contribution management and we describe the role of developer communication in each of these phases. Through a case study, we show that delayed communication can valuable cause code contributions to be abandoned.

V. CONTRIBUTIONS OF OUR DISSERTATION

Our doctoral dissertation makes a variety of contributions to the research field. In the following we highlight the key technical and conceptual contributions of each chapter.

A. Technical Contributions

The technical contributions of our dissertation focus on the development of tools and techniques with the overarching goal for mining communication data from development repositories, extracting technical information from this unstructured data, and linking communication data back to the source code of the software.

1) Chapter 3 demonstrates that communication data cannot readily be processed by traditional data mining, information retrieval and natural language processing approaches, and details common problems as well as possible technical solutions. Chapter 3 has led to the implementation of an email-mining tool named MailboxMiner\(^1\), which is publicly available and has found extensive use in the research area, e.g., the work by German et al. and Jiang et al. [10], [13].

2) Chapter 4 contributes a lightweight approach to untangling unstructured data, in order to separate technical information from natural language text, as well as a manually developed benchmark suite to evaluate and compare the performance of future approaches against.

3) Chapter 5 contributes an improved approach to link communication data to those parts of the source code that are being discussed. We demonstrate a sample application for visualizing which parts of the code are talked about the most.

4) Chapter 6 contributes a novel set of socio-technical metrics surrounding the social interactions between developers. We demonstrate that these metrics can be used to understand software quality in a more holistic way when used in conjunction with traditional source code and process based metrics.

5) Chapter 7 contributes a conceptual model of contribution management, as well as an investigation of successful practices of contribution management, which can be used by practitioners for establishing effective contribution management when moving towards an open-source business model.

B. Empirical Contributions

The empirical contributions of our dissertation are the application of these tools and techniques within a variety of case studies with the overarching goal of empirically validating our research hypothesis.

1) Chapter 3 demonstrates that communication data is unlike traditional software engineering data. Communication data exists as unstructured data that intertwines natural language text, project specific language, automated text, and technical information. We demonstrate that improper handling of this special kind of data can lead to substantial bias in data, experiments and results.

2) Chapter 5 demonstrates that different conceptual classes of links between communication data and the software can be established. We further show that the approach presented in this chapter produces a novel class of traceability links that are fundamentally different from the links established by traditional approaches.

3) Chapter 6 shows that three kinds of socio-technical relationships exist between collaboration and software quality. We demonstrate not only the existence of these relationships, but also their merits for creating defect models with explanatory power similar to traditional models based on product and process metrics. In addition, we show that a combination of these socio-technical metrics and traditional product and process metrics in defect models, yields higher

\(^1\)https://github.com/nicbet/MailboxMiner
Chapter 7 demonstrates that timely communication is a key factor of successful contribution management. Our case study on two large open source software systems, makes a strong case for the importance of effective communication in order to prevent the waste of precious resources on dead-end code contributions.

VI. SUGGESTIONS FOR EXTENSION OF OUR RESEARCH

We believe that our dissertation makes a positive contribution towards providing empirical evidence of the strong relationships between developer communication and the quality, as well as the evolution of a software product. However, we believe that our work also opens a number of research opportunities. In the following, we highlight potential future work to extend our results.

Exploration of Additional Communication Repositories

In Chapter 3, we demonstrated tools and techniques for mining communication data from email repositories. While e-mail remains the most popular means of asynchronous communication between developers, engineers use a variety of channels, such as formal weekly meetings, informal meetings like face-to-face chat in hallways and kitchens, electronic chat applications, phone, or social media [20]. As such, email repositories capture only a fraction of the communication surrounding the development process, which may contain only a partial view on the collaborative activities of developers. We believe that further exploration of tools and techniques for capturing communication through these additional channels prove valuable for obtaining a more complete and broader view on the central role of communication in software development to facilitate collaborative activities.

Understanding the significance of technical information within the communication between developers

In Chapter 4, we presented a lightweight approach for separating technical information from natural language text in communication data. We used that technical information in Chapter 5 to link communication data to the parts of the software that is being talked about. However, we believe that we need to look at a broader picture and investigate the reasons behind communicating that particular piece of information. A study by Bacchelli et al. [1] presents a first step in this direction, investigating whether software modules that are being frequently mentioned in developer communication are more likely to contain errors.

Capturing Communication Through Social Metrics

In Chapter 6, we presented a variety of socio-technical metrics surrounding the communication between developers. However, this selection of metrics is far from complete and while we demonstrate that the underlying statistical models explain software defects as well as source code metrics, the factors we captured and used as predictor variables in those models may be incomplete. To counter potential bias due to not capturing important causal factors, and to gain a better understanding of what aspects of communication influence software quality, we believe that research would greatly benefit from further investigation of qualitative aspects of communication and how we can measure these aspects from communication data.

Understanding the Semantics of SNA Metrics

We observed that SNA metrics are interpreted specific to the context of each study. However, we are not aware of any attempt to collect, summarize and put possible interpretations of the relationships between SNA metrics and software quality aspects into a common framework. We believe that further investigation of SNA metrics obtained from socio-technical networks and an attempt at generalization may provide deeper insights into the coordination of development teams, the setup of software development processes and the architectural organization of the source code.

Investigating a Broader Range of Quality Metrics

We observed that a majority of research focuses on readily available and measurable quality metrics such as the number of defects delivered to the customer, or development effort spent removing defects. However, little empirical evidence is provided by existing literature on less readily available software quality concerns, such as project health, project profit, as well as implicit quality aspects which might strongly influence future development, such as software design quality, deviations from coding standards, or documentation of the source code. We believe that further investigations on the relationships between socio-technical concerns and these largely unexplored quality aspects might yield valuable insights for practitioners and researchers alike.

Repetition of Studies

The results and recommendations reported in this study are mainly based on the observation of correlations. We would like to strongly encourage the repetition of our research on a broader range of software projects and software domains to increase generalizability of our findings.

VII. ADVICE TO A YOUNGER RESEARCHER

As part of this post-doctoral dissertation presentation, we would like to present the following pieces of advice that we would give a younger us.

- **Conferences! Conferences! Conferences!** Conferences are an essential part of your research career. They are the main forum for exchanging ideas, broadening understanding of concepts and keeping up-to-date with the state-of-the art. Additionally, conferences are a great way to “dry-run” new ideas by peers in the community and help focus research efforts into the right directions.

- **Conference Funding.** Sometimes, especially with larger labs, your thesis supervisor will not have the ability to fund your visits to all the main conferences in your research field. This is where you have to get creative to raise your own funds. Seeking out university-level travel grants, and IEEE / ACM funds can make the difference between participating in a conference, or staying home.
VIII. CONCLUSIONS

Software development is a complex symphony of a broad range of development tasks, ranging from design over documentation to the actual encoding of logic in the software’s source code. Effective software development thus requires the coordination of these activities among developers to avoid breaking the source code and introducing errors into the software.

At the heart of coordination among a group of developers stands communication between individuals, together with all the social aspects of human interaction such communication brings with it. We believe that the realization of software development as a highly social process, in which the social aspects of interactions between developers play a key factor in the quality of a software is likely to take on a central role in future software engineering research.

Our work contributes to the field of software engineering by demonstrating that software repositories contain a wealth of developer communication captured in a variety of artifacts produced throughout the software development process, which can be used by practitioners and researchers to obtain a second view on software, which is orthogonal to the traditional views that rely solely on technical facts about the source code and the process employed to create that code.

We hope that our dissertation will encourage research to explore integrating social aspects of software development in addition to traditional technical views in their analyses. Our goal is to entice practitioners to consider the impact social interactions between developers and development teams can have on the quality of their software product, with the aim to development environments that stimulate effective developer communication.

ACKNOWLEDGMENTS

The authors would like to thank Ahmed E. Hassan for creating the most wonderful, stimulating and productive research environment any doctoral student could possibly ask for. We would also like to thank our doctoral committee for all the help and guidance throughout our dissertation work, and the members of the SAIL lab for being the most amazing colleagues.

REFERENCES


